UNIVERSITA' DEGLI STUDI DI PISA

ISTITUTO DI SCIENZE DELL'INFORMAZIONE

Elementi di programmazione: sottoprogrammi e macro-definizioni

C. Montangero

Note per il corso di
TEORIA ED APPLICAZIONI DELLE MACCHINE CALCOLATRICI

Anno Accademico 1974/75

1. Introduzione

Per determinare la soluzione algoritmica di un problema complesso è, in generale, conveniente adottare la seguente metodologia:

a. scomporre il problema originale in un insieme di sottoproblemi;
b. fornire una soluzione algoritmica del problema originale, assumendo di conoscere una soluzione algoritmica per i sottoproblemi individuali;
c. affrontare singolarmente, e con la stessa metodologia, i singoli sottoproblemi, fino a ridursi a problemi immediatamente risolvibili.

Tale metodologia, per inciso, è nota come programmazione top-down (dall'alto in basso, o meglio dal generale al particolare).

I vantaggi di questo approccio sistematico sono sostanzialmente i seguenti:

1. migliore schematizzazione del problema, e migliore comprensibilità, a posteriori, della soluzione adottata;
2. facilità di individuazione di sottoproblemi che ricorrono più volte nella soluzione del problema iniziale
3. oppure di problemi già risolti - eventualmente anche in ambito diverso e/o da altri.

In pratica, per problemi non banali, non accade mai che la decomposizione avvenga in modo lineare: in generale la soluzione di un particolare sottoproblema può fornire elementi per una revisione delle scelte operate a livelli superiori. L'intero procedimento può così essere ripetuto varie volte. Ciò non inficia,comunque, l'utilità di tale approccio sistematico.

Un esempio

Ricordiamo che per peso $\pi[x]$ di una parola binaria x, si intende il numero di bit ad 1 di x. Consideriamo il

Problema 1: determinare il peso massimo, date tre parole binarie a,b e c, di 18 bit.

La fig.1 fornisce una soluzione algoritmica del problema, in cui si assume (blocchi A,B e C) di conoscere la soluzione algoritmica del seguente:

Problema 2: determinare il peso di una parola binaria x di n bit.

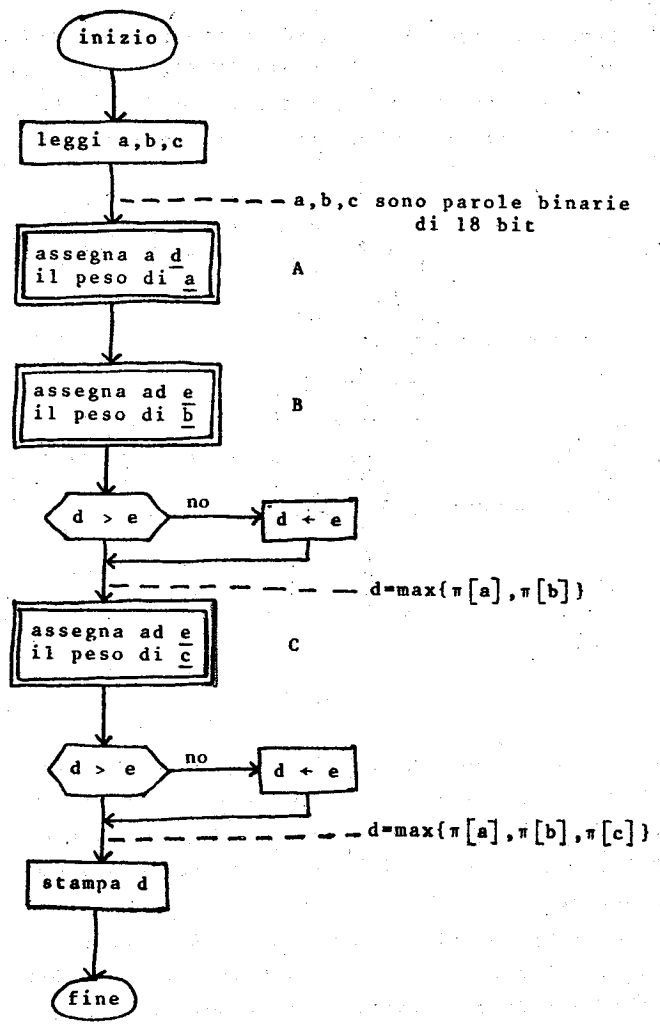


fig. 1

Tale problema viene isolato in quanto ricorre diverse volte nella soluzione di 1; la sua soluzione algoritmica - che non coinvolga alcun altro sottoproblema interessante - è data in fig.2. La parola x è considerata come un vettore di n bit, con indice variabile tra 0 e 17.

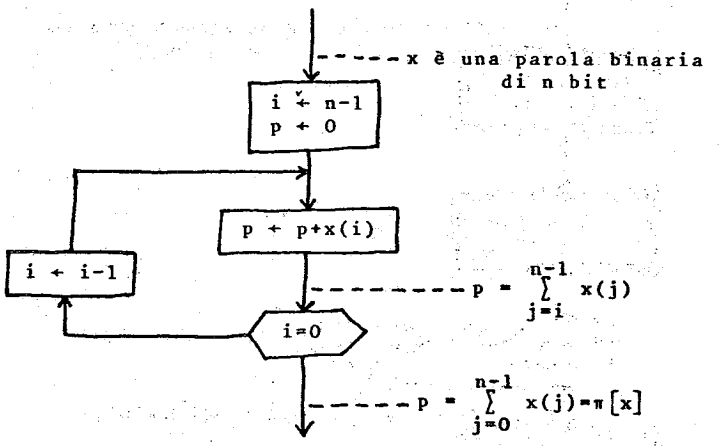


fig. 2

Pur avendo a disposizione tutti gli elementi per risolvere il problema 1, è ancora necessario evitare ambiguità e rendere così completamente determinata la soluzione, precisando le relazioni tra gli algoritmi 1 e 2, cioè le regole per la loro utilizzazione congiunta. In pratica, vengono utilizzate due tecniche:

a) macro-espansione: si utilizzano gli algoritmi 1 e 2 per definire un nuovo algoritmo, che fornisce la soluzione del problema (fig.3);
b) chiamata di sottoprogramma: la soluzione del problema è fornita dall'esecuzione congiunta degli algoritmi 1 e 2 (fig.4); le modalità con cui avviene lo scambio del controllo e delle informazioni tra i due programmi vengono precisate negli stessi algoritmi 1 e 2.

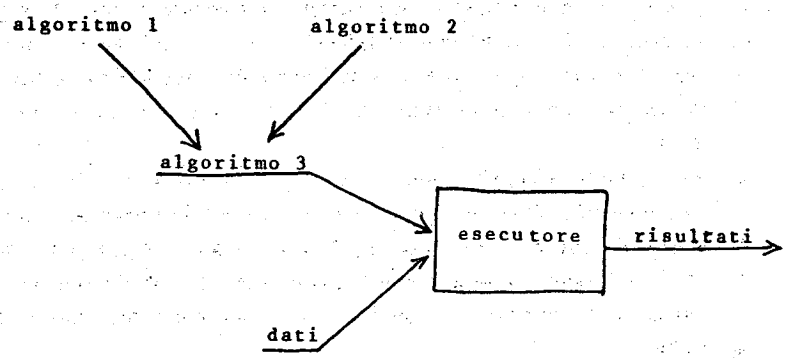


fig. 3

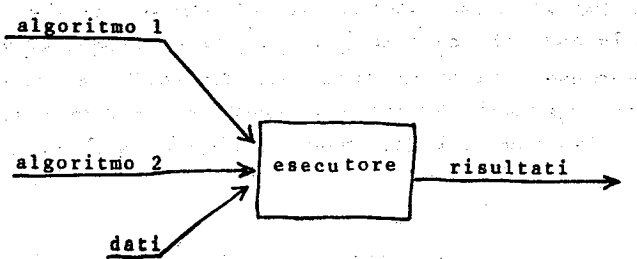


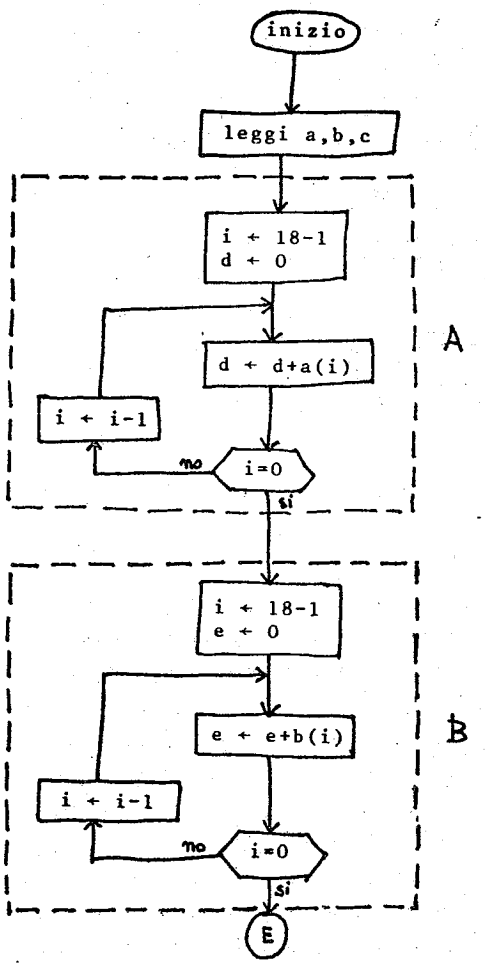
fig. 4

2. Macro-espansione

Nel caso di macro-espansione l'algoritmo risolutivo si ottiene espandendo i blocchi corrispondenti a sottoproblemi, secondo le rispettive definizioni. Questo processo è in generale realizzato automaticamente, eseguendo un opportuno algoritmo.
In questa dispensa non considereremo in dettaglio questo algoritmo: le considerazioni che seguono, tuttavia, mettono in luce i principali problemi connessi con la sua realizzazione.
Nell'esempio, i blocchi A,B e C in fig.1 sono da intendersi come abbreviazioni. L'algoritmo che risolve il problema 1 è ottenuto espandendo tali blocchi in accordo con fig.2: si veda la fig.5.
Particolare attenzione va prestata, nella espansione, alle variabili in gioco. E' conveniente introdurre alcuni termini che verranno utili nella discussione. L'algoritmo di fig.2 è una definizione di macro; le variabili che denotano dati e risultati di una macro sono i parametri formali. Ad ogni definizione di macro si associa una descrizione, contenente il nome e la lista dei parametri formali. Ad esempio, convenendo di chiamare pigreco la macro di fig.2, e notando che x, n e p sono i parametri formali, associamo alla macro la descrizione

pigreco[x,n,p].

Le variabili, come i in fig.2, necessarie per l'algoritmo definito dalla macro, che non hanno un corrispondente, dal punto di vista logico, nell'algoritmo utilizzatore, sono dette variabili locali.
I parametri formali hanno tale nome, in quanto servono esclusivamente a definire la forma della macro: al momento dell'espansione essi vengono sostituiti dai parametri attuali, che definiscono di volta in volta le informazioni su cui deve operare ogni particolare occorrenza della macro nell'algoritmo espanso. Così, ad esempio, nel caso del blocco A, x è stato sostituito con a, n con 18 e p con d, in quanto la variabile a e la costante 18 definiscono i dati per questa particolare occorrenza della macro, e la variabile d individua il supporto in cui deve essere memorizzato il risultato. Per rendere esplicita la corrispondenza tra parametri attuali e parametri formali nell'espansione, è conveniente introdurre un apposito blocco (utilizzazione di macro),



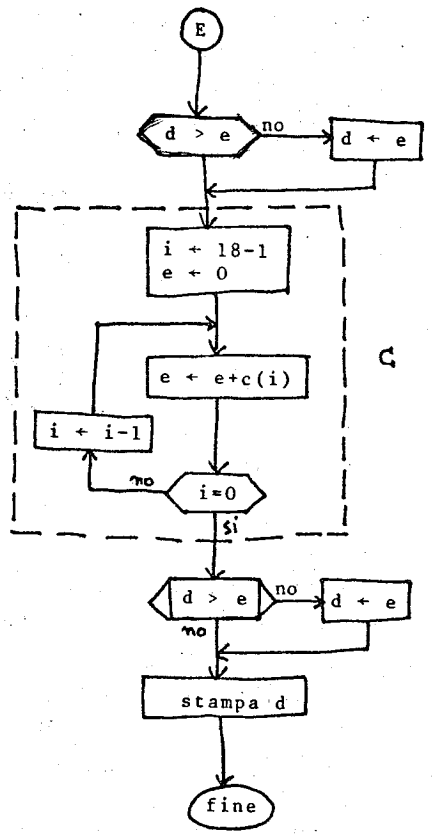


fig. 5

illustrato in fig.6. I blocchi A e B corrispondono a quelli omonimi di fig.1; essi mettono in evidenza quale macro sia da utilizzare per l'espansione (in generale si può avere più di una macro). Inoltre, la corrispondenza tra l'espressione pigreco [a,18,d] del blocco A, e la

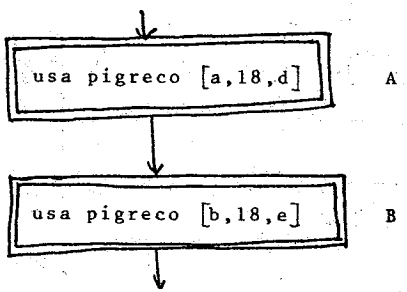


fig. 6

descrizione della macro pigreco [x,n,p], stabilisce la corrispondenza tra parametri attuali e formali: a corrisponde ad x, 18 ad n e così di seguito: in generale nell'espandere una macro, ad ogni occorrenza nella definizione del parametro formale q si sostituisce il parametro attuale che occupa, nella lista dei parametri attuali del blocco in espansione, lo stesso posto che q occupa nella descrizione della macro. Un'ultima osservazione è necessaria, relativamente alle variabili locali: esse devono essere distinte da tutte le variabili dell'algoritmo che si espande, o comunque la loro utilizzazione nella macro espansione non deve interferire con l'uso di variabili omonime nell'algoritmo da espandere. L'esempio di fig.7 rende evidenti le ragioni di questa cautela: si consideri infatti che cosa accade se la locale i della macro pigreco non viene rinominata durante l'espansione (indipendentemente dalle operazioni in P). Si ricordi che nomi di variabile diversi corrispondono a supporti di memoria diversi, mentre lo stesso nome indica lo stesso supporto.

Per concludere riassumiamo i principali aspetti dell'uso di macro definizioni:

a) corrispondenza tra parametri formali ed attuali, per effettuare correttamente l'espansione;

b) eliminazione di conflitti tra informazioni locali alla macro ed informazioni proprie dell'algoritmo da espandere.

Ogni linguaggio (cioè insieme di regole) per la descrizione di algoritmi, che preveda la possibilità di espansione di macro, deve stabilire delle regole per la definizione, la descrizione ed utilizzazione di macro, e convenzioni sulla natura dei parametri attuali. Ad esempio, in questo paragrafo abbiamo considerato come parametri attuali solo variabili e costanti. Ci limitiamo qui ad accennare al fatto che espressioni aritmetiche, nomi di macro ecc., possono utilmente svolgere il ruolo di parametri attuali. Le convenzioni succitate costituiscono le prescrizioni per l'algoritmo di espansione.

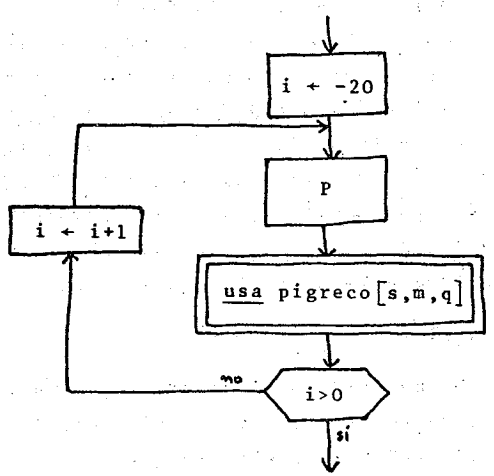


fig. 7

3. Chiamata di sottoprogramma

In questo caso i programmi restano distinti e collaborano alla soluzione del problema scambiandosi sia il controllo dell'esecuzione che informazioni su cui operare, secondo una ben definita relazione di dipendenza: un programma (programma chiamante) invoca l'esecuzione di un altro programma (il sottoprogramma) per utilizzarne i risultati,

trasferendogli i dati su cui operare. L'esecuzione del programma chiamante viene ripresa al termine dell'esecuzione del sottoprogramma, dal punto in cui è stata sospesa per la chiamata (ritorno dal sottoprogramma). Precise convenzioni stabiliscono la modalità di restituzione al programma chiamante dei risultati del sottoprogramma. E' importante osservare subito che il programma chiamante può essere a sua volta un sottoprogramma: un sottoprogramma può chiamarne un altro, in perfetto accordo con la metodologia illustrata nel primo paragrafo. Un solo programma non è chiamato da altri, ed è detto programma principale.

Nell'esempio di figg.1 e 2, è possibile organizzare l'algoritmo di fig.2 come un sottoprogramma, chiamato ripetutamente dall'algoritmo 1, in corrispondenza ai blocchi A, B e C. Tale situazione è illustrata in fig.8, per la cui comprensione sono necessarie alcune osservazioni:

a) valgono anche nel caso di sottoprogrammi, le definizioni, date nel paragrafo precedente, di parametri formali, parametri attuali e variabili locali. E' pure conveniente associare ad ogni sottoprogramma una descrizione, composta da nome e lista dei parametri formali. Nel seguito useremo pigreco [x,n,q] come descrizione del sottoprogramma di fig.2.

b) I blocchi A,B e C, di fig.8 sono blocchi di chiamata, e mettono in evidenza il nome del sottoprogramma ed i parametri attuali.

c) I blocchi tratteggiati D ed E indicano estensioni dell'algoritmo di fig.2, per organizzarlo come sottoprogramma.

d) Scopo del blocco D è l'associazione tra parametri formali ed attuali: ad esempio, in corrispondenza alla chiamata A, x deve essere associato ad a, n a 18 e p a d, così che al termine dell'esecuzione del sottoprogramma, in d sia memorizzato il peso della parola a. Per la chiamata B, l'associazione deve avvenire tra x,n,p e b,18,e rispettivamente, così da memorizzare in e il peso di b. Analogo discorso vale per la chiamata C. Il meccanismo con cui l'associazione avviene, come pure la natura dei parametri attuali, dipende dal linguaggio utilizzato: in questa dispensa prenderemo in considerazione solo il caso del linguaggio macchina (vedi par.4). In ogni caso notiamo che

programma principale

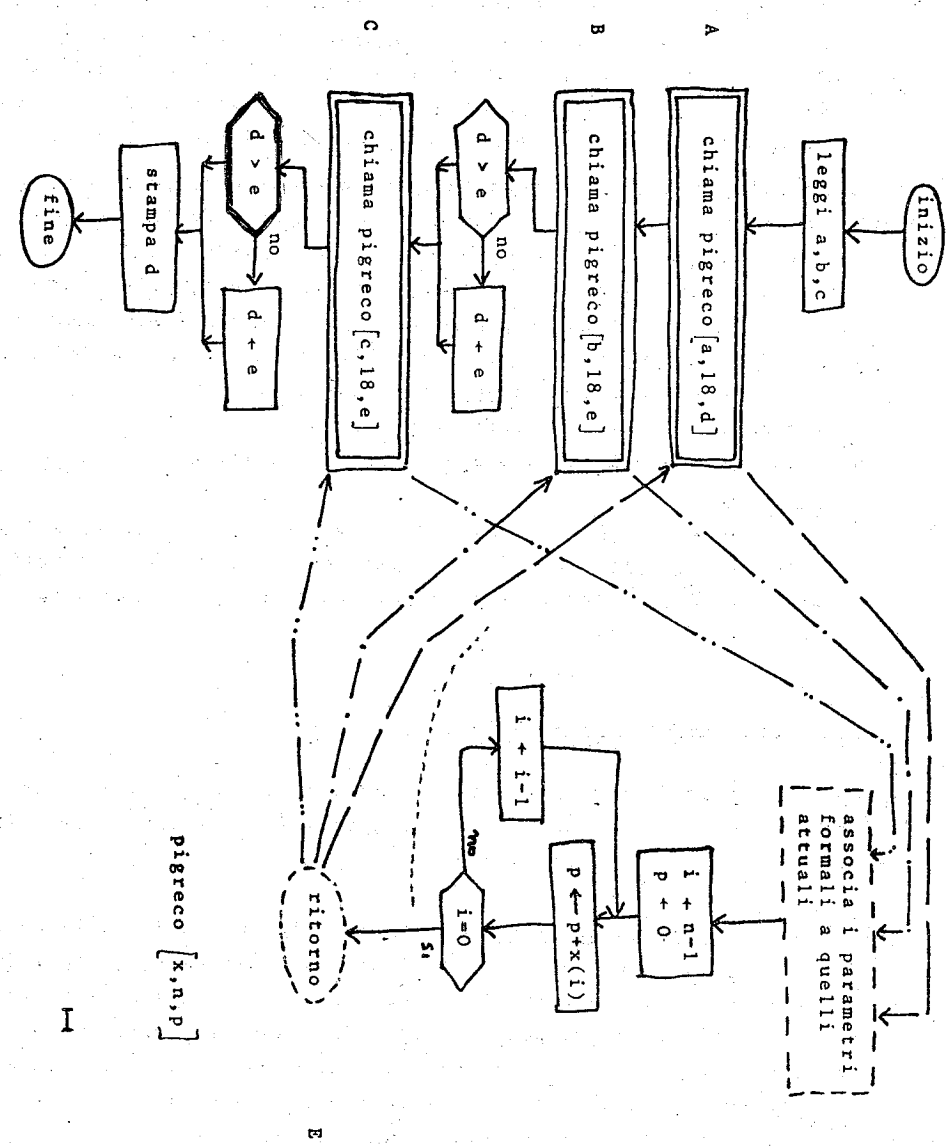


fig. 8

tale meccanismo deve mettere in comunicazione gli spazi di memoria del programma chiamante e del sottoprogramma, che sono distinti. Tale distinzione comporta che non vi siano problemi relativi alle variabili locali: variabili omonime in programmi distinti denotano supporti di memoria distinti, salvo esplicita convenzione contraria.

e) Scopo del blocco E è di restituire il controllo al programma chiamante, al termine dell'esecuzione del sottoprogramma: le coppie di frecce tratteggiate che collegano PC* e SP in fig.8 mettono in evidenza che il sottoprogramma ritorna al programma chiamante, in modo che la esecuzione di quest'ultimo riprenda dal blocco successivo a quello di chiamata. Per concludere, i blocchi di chiamata hanno un significato operativo, e definiscono le seguenti operazioni:

a) trasmissione dei parametri attuali;
b) trasmissione dell'informazione per il ritorno (punto di continuazione);
c) cessione del controllo dell'esecuzione al sottoprogramma. Questo, a sua volta, provvede all'associazione tra parametri formali ed attuali ed al ritorno al programma chiamante, nel punto da cui è stato chiamato.

4. Confronto fra macro e sottoprogrammi

In questo paragrafo confronteremo brevemente le due organizzazioni, relativamente ad alcuni parametri: occupazione di memoria, tempo di esecuzione e costo di utilizzazione.

4.1 - Per occupazione di memoria intendiamo il numero di celle occupate dal programma che codifica, in linguaggio macchina, l'algoritmo A. Supponiamo che l'algoritmo A si ottenga dall'utilizzazione congiunta degli algoritmi (corrispondenti ad altrettanti sottoproblemi) $A_1,\ldots,A_n$. Indicheremo con σ_i l'occupazione di memoria per l'algoritmo A_i; σ_i non comprende le celle necessarie per eventuali blocchi di uso di macro o di chiamata di sottoprogramma, di cui terremo conto a parte. Supporremo che A_1 sia il programma principale, e che il blocco di

* - PC ed SP indicano rispettivamente programma chiamante e sottoprogramma.

uso dell'algoritmo A_i compaia m_i volte ($m_1=1$). Per l'esempio dei paragrafi precedenti, la fig.1 descrive l'algoritmo A_1, la fig.2 A_2 con $m_2=3$.

Nel caso di macro-espansione, l'occupazione di memoria per l'algoritmo A è

$$O_M = \sum_{i=1}^{n} m_i \sigma_i \ .$$

Per l'esempio precedente: $O'_M = \sigma_1 + 3\sigma_2$.

Un sottoprogramma, invece, viene memorizzato una volta sola: non è però vero che l'occupazione complessiva sia data solo da

$$\sum_{i=1}^{n} \sigma_i \ .$$

E' infatti necessario tener conto che:

a) al programma chiamante deve essere aggiunto il codice relativo alla trasmissione dei parametri e del controllo (sequenza di chiamata). Indicheremo con c_i l'occupazione di memoria della sequenza di chiamata di A_i ($c_1=0$);
b) alla codifica di A_i deve essere aggiunto il codice per l'associazione dei parametri e per il ritorno; sia a_i la relativa occupazione di memoria ($a_1=0$).

Abbiamo quindi, per il caso di organizzazione a sottoprogrammi,

$$O_S = \sum_{i=1}^{n} (c_i\, m_i + a_i + \sigma_i).$$

Per confrontare le due organizzazioni, consideriamo il rapporto tra il contributo, di un singolo algoritmo A_i, all'occupazione di memoria nel caso di sottoprogramma e di macro:

$$\eta_O = \frac{c_i m_i + a_i + \sigma_i}{m_i \sigma_i} = \frac{c_i}{\sigma_i} + \frac{1}{m_i}\frac{a_i+\sigma_i}{\sigma_i} \ .$$

L'organizzazione a sottoprogrammi è vantaggiosa rispetto all'occupazione di memoria, se $\eta_O<1$. Perchè ciò si verifichi è necessario che $c_i<\sigma_i$: l'occupazione dovuta alla sequenza di chiamata deve essere (molto) minore di quella dell'algoritmo. Il secondo termine in η_O

esprime semplicemente il fatto che l'occupazione di memoria destinata all'associazione ed al ritorno, è tanto meno rilevante quanto maggiore è il numero di chiamate del sottoprogramma.

4.2 - Per confrontare le due organizzazione relativamente al tempo di esecuzione, indicheremo con t_{M_i} il tempo di esecuzione della macro A_i; t_{M_i} è una funzione dei dati, in generale. A parità di dati, il sottoprogramma corrispondente ad A_i verrà eseguito in un tempo $t_{S_i} = t_{M_i} + t_{c_i} + t_{a_i}$, dove t_{c_i} tiene conto della sequenza di chiamata, e t_{a_i} tiene conto dell'associazione tra parametri formali ed attuali. Il rapporto

$$\eta_T = \frac{t_{S_i}}{t_{M_i}} = 1 + \frac{t_{c_i} + t_{a_i}}{t_{M_i}}$$

esprime il fatto (ovvio) che l'organizzazione a sottoprogrammi è tanto peggiore, rispetto a quella a macro e relativamente al tempo di esecuzione, quanto più tempo si spende per la chiamata e l'associazione dei parametri.

4.3 - Per costo di utilizzazione, intendiamo il costo delle operazioni necessarie per giungere all'esecuzione del programma, supponendo di voler utilizzare "a scatola chiusa" un algoritmo di cui non si conosca la struttura, ma solo la descrizione funzionale e le modalità d'uso e di cui si abbia già la codifica. Nel caso di sottoprogramma, il costo di utilizzazione è dato dalla codifica delle chiamate e dal caricamento del sottoprogramma. Nel caso di macro, ogni espansione richiede, in generale, l'esame di tutta la macro: operazione che è, in generale, più costosa di quelle richieste per la chiamata del corrispondente programma.

La contradditorietà dei risultati del confronto tra le due organizzazioni spiega il fatto che nessuna delle due abbia, in pratica, prevalso sull'altra e che, invece, i linguaggi più evoluti per la codifica di algoritmi prevedano entrambe le possibilità, per dar modo al programmatore di utilizzare quella più conveniente, caso per caso.

5. Sottoprogrammi in linguaggio macchina

Le operazioni necessarie per l'esecuzione di un sottoprogramma sono le seguenti:

1. cessione del controllo dal PC al SP
2. trasmissione al SP dell'informazione per il ritorno
3. trasmissione dei parametri attuali
4. associazione tra parametri formali e attuali
5. ritorno al PC
6. salvataggio e ripristino dei registri utilizzati dal SP.

Il senso dell'ultima operazione, che non era stata considerata, a differenza delle altre, nella discussione generale del par.3, sarà chiarito dalle considerazioni successive.

5.1 - Salto a SP e ritorno

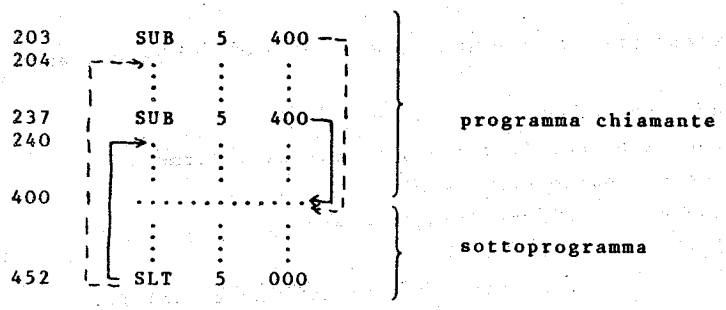
Il corredo di istruzioni di ogni calcolatore comprende, in generale, una istruzione di salto a sottoprogramma, che risolve i problemi connessi con le operazioni 1 e 5. Essa consiste in un salto incondizionato alla prima istruzione del SP, preceduto dalla memorizzazione, in un opportuno supporto, dell'indirizzo di ritorno, cioè dell'istruzione da cui riprende l'esecuzione del PC al termine del SP. Nel CANE [1], questa istruzione ha codice operativo 41 (mnemonico SUB) utilizza un registro indice per memorizzare l'indirizzo di ritorno. L'istruzione di salto a SP ha il formato SUB J I; il suo effetto e, simbolicamente(*):

$$(J \neq 0) \Rightarrow (rX[J] \leftarrow rCP \quad rCP \leftarrow I)$$

L'indirizzo successivo a quello dell'istruzione SUB viene memorizzato in rX[J]; l'istruzione successiva è prelevata dalla cella di indirizzo I. Il registro J, specificato dall'istruzione SUB, è detto registro di chiamata, e contiene l'informazione per il ritorno. Si considerino ad

(*) Si assume che il registro contatore programma rCP venga incrementato nella fase interpretativa.

esempio le istruzioni:



L'esecuzione dell'istruzione in 203 provoca un salto al sottoprogramma con rX[5]≡204. Purchè il SP non alteri questo registro, l'ultima istruzione eseguita dal SP, il salto di ritorno in 452, riattiva il PC dall'istruzione successiva alla chiamata, in 204. Alla seconda chiamata, rX[5]≡240, per cui il ritorno dal SP avviene all'istruzione in 240.

Riassumendo, il collegamento di PC e SP, per ciò che riguarda il controllo, si effettua tramite una istruzione di salto a SP, il cui uso richiede una convenzione comune ai due programmi, relativamente

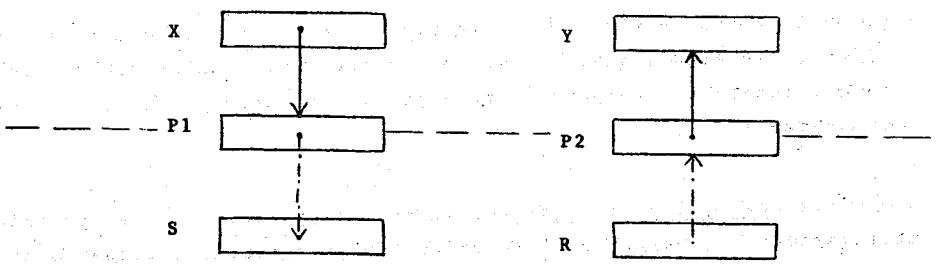
1. alla collocazione in memoria del SP, che definisce l'indirizzo di chiamata;
2. al registro di chiamata, cioè il supporto di memoria che permette di effettuare correttamente il ritorno.

5.2 - Trasmissione ed associazione dei parametri

Anche per la trasmissione dei dati al SP dei risultati al PC sono necessarie delle convenzioni, relative a:

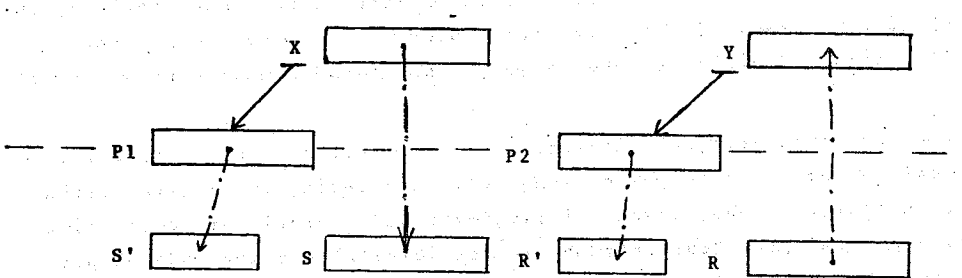
1. i supporti di memoria utilizzati nella comunicazione;
2. le modalità di utilizzo di questi supporti.

Per quanto riguarda quali supporti utilizzare, la scelta è tra registri e celle di memoria. Per le modalità di utilizzo dei supporti convenzionalmente in comune, considereremo due casi: trasmissione per valore (by value) e per indirizzo (by reference). Nel primo caso, il supporto comune è utilizzato per trasmettere direttamente il dato o il risultato. Nel secondo caso, viene trasmesso l'indirizzo del parametro attuale: il sottoprogramma può così accedere alla memoria dedicata al PC; questo accesso è invece impossibile nel caso di trasmissione per valore. Le fig.9 e 10 illustrano le due possibilità: si assume un SP con due parametri formali, S in ingresso ed R in uscita. La linea a tratto separa le aree di memoria dedicate al PC (in alto) ed al SP (in basso): P1 e P2 denotano i supporti comuni per la comunicazione tra i programmi. Le frecce continue indicano le azioni del PC per la trasmissione dei parametri attuali X ed Y, quelle a tratto e punto le azioni del SP per l'associazione con i formali. Non sempre, tutti i supporti indicati in fig.9 sono fisicamente, oltre che logicamente, distinti: in alcuni casi, come vedremo nel seguito, essi possono coincidere, almeno in parte, con conseguente risparmio sia in occupazione di memoria che in tempo di esecuzione.



trasmissione per valore

Fig. 9



trasmissione per indirizzo

Fig. 10

5.3 - Un programma per il problema 2

Consideriamo una semplificazione del problema 2, assumendo che n=18. L'algoritmo di fig.11 risolve il problema; nella sua traduzione in programma CANE in fig.11 b) si è assunto che la parola x, di cui si vuole determinare il peso, sia disponibile nel registro rB.

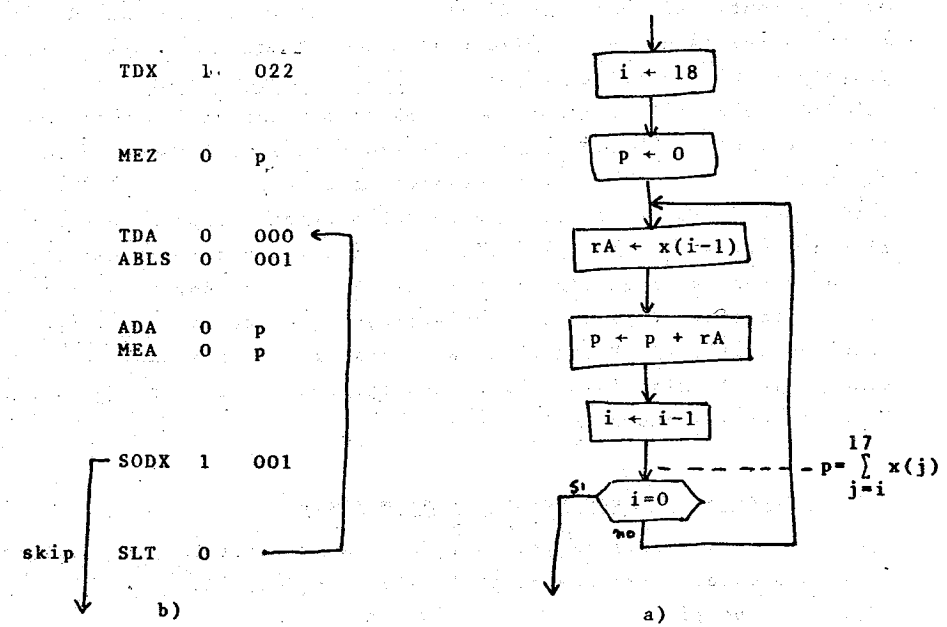


Fig. 11

L'indirizzo corrispondente a p verrà specificato nel seguito. Per organizzare questa sequenza di istruzioni come un sottoprogramma è necessario specificare le convenzioni per la comunicazione tra PC e SP. In tutti gli esempi successivi assumeremo:

rX[6] come registro di chiamata,
400 come indirizzo di chiamata,

e che alle variabili a,b,c,d,e di fig.1, corrispondano gli indirizzi da 073 a 077, relativi alla prima istruzione del PC.

5.4 - Trasmissione dei parametri attraverso registri programmabili

Come esempio di trasmissione attraverso registri programmabili, assumiamo la seguente convenzione:
sia p che x per valore attraverso rB.
La sequenza di chiamata sarà allora:

```
   ⋮
TRB  0  073        copia di a per il sottoprogramma
SUB  6  400
```

Al ritorno, il risultato può essere memorizzato in d, con

```
MEB  0  076.
```

Per organizzare la sequenza di istruzioni di fig.11 b) come sottoprogramma, in base alle convenzioni precedenti, è sufficiente premettervi le istruzioni per il salvataggio dei registri utilizzati come locali, in due locazioni proprie del SP, che indichiamo simbolicamente con rX1 ed rA:

```
MEX  1  rX1
MEA  0  rA.
```

Per la restituzione del parametro di uscita ed il ritorno, preceduto dal ripristino dei registri modificati dal SP, bisogna aggiungere in coda le istruzioni

```
TRB  0   p
TRX  1  rX1
TRA  0  rA
SLT  6  000.
```

La convenienza di salvare e ripristinare i registri programmabili utilizzati come locali e modificati dal SP è ovvia: il PC può assumere che il SP non alteri i registri, di cui ha quindi la completa disponibilità.
Il sottoprogramma risultante, in fig.12, è scritto in un formato che differisce da quello del caricatore rilocante solo per la presenza dei codici mnemonici al posto di quelli operativi. Il SP occupa 21_8 celle di memoria, di cui solo 16_8 devono essere predisposte dal caricatore, a partire dalla locazione 400.

```
000     MEX    1   016*   }salv.reg.
        MEA    0   017*
        TDX    1   022
        MEZ    0   020*
004     TDA    0   000
        ABLS   0   001
        ADA    0   020*
        MEA    0   020*
        SODX   1   001
        SLT    0   004*
        TRB    0   020*     restituzione p
        TRX    1   016*   }ripristino reg.
        TRA    0   017*   }
        SLT    6   000      ritorno
016                         rX1
017                         rA
020                         p
```

Fig. 12

5.5 - Discussione dell'esempio

L'utilizzazione dei registri per la comunicazione tra PC e SP è efficiente e semplice. Essa è conveniente se i parametri sono in numero limitato, ad esempio quando il sottoprogramma calcola una funzione di poche variabili, come nell'esempio precedente. Un altro caso è quello di SP sviluppati nell'ambito di un singolo progetto. In entrambi i casi gli svantaggi della trasmissione tramite registri, sono poco sentiti; tali svantaggi sono: il limitato numero di registri disponibili e i vincoli così imposti al PC nell'uso dei registri stessi. Le conseguenze del primo svantaggio sono ovvie. Il fatto che il SP sia sviluppato nell'ambito di un particolare progetto implica che la scelta sull'utilizzazione dei registri, venga fatta globalmente, e quindi, in generale, in modo da non provocare conflitti nell'uso dei registri stessi da parte di SP diversi. Ciò non è affatto garantito, invece per SP sviluppati per uso generale, da parte di una comunità di utenti. In questo caso un programma che utilizzi più SP non ha più la libera disponibilità dei registri programmabili: può trovarsi di fronte a convenzioni in conflitto che lo costringono a salvare il contenuto di alcuni registri e ripristinarlo dopo ogni chiamata. La trasmissione attraverso registri, in generale, vincola il PC, in quanto aumentano le assunzioni che il PC deve rispettare per utilizzare il SP. E' quin-

di conveniente, per applicazioni di uso generale, con un numero di parametri superiore alle poche unità, utilizzare celle di memoria per la trasmissione dei parametri, e lasciare al SP il compito di salvare e ripristinare i registri che utilizza. In questo modo vengono ridotti i vincoli imposti al PC per l'utilizzazione del SP. Sempre per evitare conflitti, nel caso di SP di uso generale, conviene lasciare al PC la scelta di quali celle utilizzare ad ogni chiamata: utilizzando un gruppo di celle contigue (vettore di trasferimento v.d.t.) è sufficiente che il PC comunichi al SP l'indirizzo della prima di tali celle, perchè il SP sia in grado di provvedere all'associazione tra parametri formali ed attuali. Questo può essere fatto, nel CANE, memorizzando (durante il caricamento del programma oppure in esecuzione) il puntatore al v.d.t. (cioè l'indirizzo del suo primo elemento) nella cella successiva a quella di chiamata. Il ritorno dovrà avvenire quindi alla seconda cella dopo la chiamata, ed il registro di chiamata rende possibile al SP l'accesso ai parametri. Come esempio, nel prossimo paragrafo riscriviamo il sottoprogramma corrispondente alla fig. 11 a) in base a nuove convenzioni.

5.6 - Trasmissione mediante celle di memoria

Assumiamo le seguenti convenzioni, per la trasmissione dei parametri:

- x per valore, attraverso il 1° elemento del v.d.t.
- p per indirizzo, " " 2° " " "

Alla sequenza di fig.11 b) è necessario premettere le istruzioni necessarie per accedere al v.d.t.:

```
MEX   6   esg
ESG   0   esg          -          (1)
```

La cella il cui indirizzo è indicato simbolicamente con esg deve contenere, dopo il caricamento,

```
TRX   5   000
```

In seguito alla memorizzazione del registro di chiamata in esg, l'esecuzione della seconda delle (1) provoca, in definitiva, il trasferimento in rX[5] del puntatore al v.d.t. Facendo seguire alle (1) la istruzione

```
TRB   5   000
```

siamo in grado di eseguire la sequenza 11 b).

Per la restituzione del risultato, è sufficiente aggiungere le istruzioni:

```
        ADDX    5   001     rX[5]← indirizzo 2° elemento v.d.t.
        MEX     5   esg
        ESG     0   esg     rX[5]← indirizzo in cui memorizzare il
                                   risultato
        MEA     5   000     restituzione del risultato.
```

Il sottoprogramma completo delle istruzioni per il salvataggio e ripristino dei registri e del ritorno è dato in fig.13.

```
000     MEX     5   031*  }
        MEX     1   032*  }  salvataggio registri
        MEA     0   033*  }
        MEB     0   034*  }
        MEA     6   030*  }
        ESG     0   030*  }  determinazione v.d.t.
        TRB     5   000       predisposizione 1° parametro
         :      :    :
         :      :    :         istruzioni di fig.11 b) con p≡035*
         :      :    :
017     ADDX    5   001
        MEX     5   030*
        ESG     0   030*
        MEA     5   000       restituzione 2° parametro
        TRX     5   031*  }
        TRX     1   032*  }
        TRA     0   033*  }  ripristino registri
        TRB     0   034*  }
        SLT     6   001       ritorno
030     TDX     5   000       esg
031                           rX[5]
032                           rX[1]
033                           rA
034                           rB
035                           p
```

Fig. 13

Per completare la discussione consideriamo il programma chiamante, corrispondente all'algoritmo di fig.1, in cui si sia assunto n=18, così da utilizzare il SP di fig.13.
Supponendo di utilizzare il software di servizio del CANE, il pacco

programma inizierà con la scheda sterisco, seguita dalle

```
* CAR 315
 :  :  :
 :  :  :      ingresso dati
 :  :  :
```

Per la prima chiamata del sottoprogramma si utilizzano le celle di indirizzo relativo 100, 101 come v.d.t.; per le successive, le celle 102, 103. La predisposizione del 2° parametro, per ogni chiamata, è fatta al caricamento. Seguono l'ingresso le prime due chiamate del SP:

```
        TRA     0   073*
        MEA     0   100*
        SUB     6   400       1ª chiamata, per determinare π[a]
        NOP     0   100*      indirizzo v.d.t.
        TRA     0   074*      1° ritorno
        MEA     0   102*
        SUB     6   400       2ª chiamata, per determinare π[b]
        NOP     0   102*      indirizzo v.d.t.
```

Segue una sequenza di istruzioni che lascia in rA il maggiore tra d ed e:

```
        TRA     0   076*      2° ritorno
        CFA     0   077*
        SMAG    0             salto per evitare l'istruzione successi-
        TRA     0   077*                              va se d>e
```

Segue l'ultima chiamata del SP:

```
        TRB     0   075*
        MEB     0   102*      predisposizione parametro
        SUB     6   400
        NOP     0   102*      indirizzo v.d.t.
```

Con riferimento alle fig.9 e 10 si hanno le seguenti corrispondenze, relative a quest'ultima chiamata:

per il parametro x, con la fig.9:

X≡075* P1≡102* S≡rB

per il parametro p, con la fig.10:

Y≡077* P2≡103* R'≡rX[5] R≡rA

La sequenza di istruzioni del PC prosegue con l'ultimo test:

```
CFA    0   077*     3° ritorno
SMAG   0             salto per evitare la prossima istruzione,
TRA    0   077*                              se d>e
```

Le ultime istruzioni, relative alla stampa, sono seguite dal salto al nucleo di controllo e dalla predisposizione del 2° parametro attuale nei v.d.t.:

```
        :   :   :     stampa
       SLT  0   006   alt
 101   NOP  0   076*
 103   NOP  0   077*
```

Si ricordi che l'indicazione esplicita dell'indirizzo modifica la sequenza di caricamento([1]pag. 32). Le ultime schede provocano il caricamento del sottoprogramma e l'esecuzione del programma; la scheda *ALT che segue i dati indica la fine del lavoro:

```
*CAR   400
  :    :  :     sottoprogramma di fig.13
*ESG   315
  :    :  :     dati, nel formato richiesto dal programma
*ALT
```

Bibliografia

1. A.Grasselli e G.Pacini - Il calcolatore didattico "CANE". Manuale di riferimento. Nota didattica ISI 1974/75.

2. D.Knuth - The art of programming. Vol.I. Addison Wesley.